NPS55GD 8051A

UNITED STATES
NAVAL POSTGRADUATE SCHOOL

ON SOLVING INTEGER PROGRAMS

by

Harold Greenberg

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Robert W. McNitt, USN
Superintendent

R. F. Rinehart
Academic Dean

ABSTRACT:

This report contains new methods of finding integer solutions to linear programming problems. The approaches presented, with illustrative examples, emphasize the use of dynamic programming techniques.

In addition, a new branching scheme is presented that is a natural extension of linear programming methods.

Department of Operations Analysis

NPS55GD 8051A

May 1968

TABLE OF CONTENTS

Introduction

This report is a collection of algorithms developed by the author for the solution of integer programs. In Section 1, an algorithm for computing a one-dimensional knapsack function is presented.

Section 2 consists of an algorithm for the knapsack problem where the constraint equation is a congruence. By first transforming the linear programming solution of the integer program into an equivalent knapsack problem, the integer solution may often be found by rounding the continuous solution.

Section 3 consists of an algorithm for the general solution of integer linear programs. The continuous linear programming solution is found first. If the continuous solution is fractional, then a linear congruence is added as a constraint. A dynamic programming formulation of the problem is made and the optimal integer solution is found.

The final algorithm in Section 4 is a branching algorithm, which produces an optimal integer solution to the problem by systematically changing the bounds on the variables. The method requires minimum excess computer storage over that required to solve the linear programming problem. In addition, dynamic

programming enumeration techniques, similar to those used in the previous algorithm, may be used to speed the calculations once a solution to a linear program is obtained.

1. An Algorithm for the Computation of Knapsack Functions

In this section, we present an algorithm for computing a one-dimensional knapsack function defined by:

$$F(x) = \max \left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} a_j x_j = x,\ x_j \geq 0,\ x_j \text{ integer} \right\}$$

where the c_j and the a_j are given constants.

Knapsack problems have been studied in [5], [6], [7], [12]. See these references for physical applications. In this section we present an algorithm that is more general and simpler than that given in [7]. Both are modified dynamic programming algorithms. Ours, however, has the computational advantage of being a single-pass algorithm, while the algorithm in [7] requires the maintenance of a length grid to permit backtracking to find the values of the variables.

A modified dynamic programming algorithm is also given in [12]. However, it is presented in a network context of considerable complexity and appears to have no computational advantage over [7].

It is evident from the work presented here that the problem can be solved as a relatively simple enumeration with a very simple algorithm. The enumeration is for only feasible values and, thus, represents an additional computational advantage over that in [7] or [12]. The greater generality found here is that the constraint can be an equality constraint with no restrictions on the constants; the algorithms in [7] and [12] can solve equality constraint problems provided there is a variable with unit coefficient in the equality.

We define the knapsack problem as: find x_j, $j=1,\ldots,n$ that

$$\begin{aligned} &\text{maximizes} \quad \sum_{j=1}^{n} c_j x_j \\ &\text{when} \quad \sum_{j=1}^{n} a_j x_j = L \\ &x_j \geq 0,\ x_j \text{ integer}, \end{aligned} \tag{1}$$

where each c_j is a positive number, each a_j is a positive integer, and L is a feasible positive integer. In the usual knapsack problem the constraint in (1) is an inequality constraint. We consider the more general problem with the equality constraint. In problems with an inequality we would simply have a slack variable.

The one-dimensional knapsack function is defined from (1) as:

$$F(x) = \max \left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} a_j x_j = x,\ x_j \geq 0,\ x_j \text{ integer} \right\}. \tag{2}$$

If $F(x)$ can be found for all values of x and in turn the x_j be found that produces the solutions, then (1) is solved when $F(L)$ is determined.

Following [7], the knapsack function can be written as

$$F(x) = \max_{j:a_j \leq x} [c_j + F(x - a_j)] \tag{3}$$

$$F(0) = 0,$$

for values of x that are feasible. In the algorithm to follow we will use only feasible values of x; thus we can write (3) in a simpler form than in [7].

One immediate solution to (3) can be found by finding $a_r = \min a_j$, for all j. This produces $F(a_r) = c_r$. We then replace x by $x - a_r$ in (3), and we obtain

$$F(x - a_r) = \max_{j:a_r+a_j \leq x} [c_j + F(x-a_r-a_j)], \tag{4}$$

which we substitute for the $F(x - a_r)$ term on the right side of (3). We then can produce another immediate solution. We continue this way until $F(L)$ is found. We also keep track of which j produces the solution to determine the optimal x_j at the end. We also only consider distinct a_j values. If $a_j = a_k$ and $c_j \geq c_k$, $j \neq k$, then take $x_k = 0$. This entire procedure is contained in the following algorithm:

1. List the values of the problem as follows:

1	2	3	...	n
c_1	c_2	c_3	...	c_n
a_1	a_2	a_3	...	a_n
$x_j=0$				

Set m=0. Go to 2.

2. Given the list, find $a_r = \min_{a_j>m} a_j$ for columns in all sections.

If $a_r = L$, go to 4. Otherwise set $m = a_r$ and go to 3.

3. Add a new section of columns to the list, if possible, as follows:

Calculate $a'_t = a_r + a_t$ and $c'_t = c_r + c_t$ for t=1, ..., n.

Add a column headed by t if:

a) $a'_t \leq L$.

b) a'_t is not on the list.

c) a'_t is on the list and has a corresponding c_t value that is smaller than c'_t.

d) Underneath the section added write the x_j values from the section where $m = a_r$ was found. Increase x_r by one for the new section.

Sections after the first may be deleted after they are no longer needed.

Go to 2.

4. The problem is solved with solution c_r. The values of the variables are found below the section where $a_r = L$ appears; increase x_r by one.

This completes the algorithm. The algorithm may be modified as in [7] if L is much larger than the a_j. The algorithm may be modified appropriately to handle the discounted problem [12], where α^{a_j} multiplies $F(x-a_j)$ on the right hand side of (3).

Example:

$$\text{Max } 2x_1 + \frac{10}{3}x_2 + 5x_3$$

$$\text{when } 2x_1 + 3x_2 + 4x_3 = 6.$$

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad \text{all integer.}$$

We list

1	2	3
2	$\frac{10}{3}$	5
2	3	4
$x_j=0$		

(5)

and min $a_j=2$. We add the section

2	3
$\frac{16}{3}$	7
5	6
$x_1=1$	

(6)

We obtain $\min a_j = 3$ in (5). No new section is added.

We obtain $\min a_j = 4$ in (5). No new section is added.

The solution is now evident in (6), where $a_j = 6$. We thus have

$F(6) = 7$, $x_1 = 1$, $x_3 = 1$ as optimal.

2. Knapsack Solutions in Integer Programming

In this section, we present an algorithm for solution to the problem:

$$\text{minimize} \sum_{i=1}^{m} c_j x_j \quad \text{when} \sum_{j=1}^{m} a_j x_j \equiv b \pmod{D}.$$

This problem solution is useful for solving integer programs.

We are interested in the integer programming problem:

find $X=(x_1, x_2, \ldots, x_n)$ that

$$\begin{aligned} &\text{minimizes } CX \\ &\text{when } \quad AX = B \qquad (7) \\ &\qquad x_j \geq 0 \text{ and integer, } j=1, \ldots, n, \end{aligned}$$

where C is an n-vector, B is an m-vector, and A is an mxn matrix. We assume that the components of C, B, and the elements of A are integers.

Gomory [8] has presented a method that may solve (7) under certain circumstances. He solves an equivalent knapsack problem by dynamic programming to round off the linear programming solution to (7). Shapiro [11] presents another dynamic programming algorithm.

In this section, we present a simple algorithm that solves the equivalent knapsack problem. We also present a simpler equivalent knapsack problem than in [8] or [11].

The linear programming solution to (7) transforms (7) to

$$\begin{aligned} &\text{minimize} \quad d + \sum_{j\in\bar{G}} c_j x_j \\ &\text{when} \quad X_G + \sum_{j\in\bar{G}} \alpha_j x_j = \alpha_0 \qquad (8) \\ &\qquad x_j \geq 0 \text{ and integer}, \quad j=1,\ldots,n, \end{aligned}$$

where $X_G = \{x_i \mid i\in G\}$, $\alpha_0 \geq 0$, $c_j \geq 0$, G is the set of indices of the basic variables and $\bar{G}$ is the set of indices of the non-basic variables. The vectors α_j ($j=0$ and $j\in\bar{G}$) are column vectors. If α_0 is all integer then $x_j=0$ ($j\in\bar{G}$), $X_G = \alpha_0$ is an optimal integer solution. If any of the α_0 are fractional, then the problem (8) is reduced to an equivalent knapsack problem

$$\begin{aligned} &\text{minimize} \quad \sum_{j\in\bar{G}} c_j x_j \\ &\text{when} \quad \sum_{j\in\bar{G}} a_j x_j \equiv b \pmod{D} \qquad (9) \\ &\qquad x_j \geq 0 \text{ and integer}, \end{aligned}$$

where the constraint in (9) is a single equation (in this case a congruence) and D is the determinant of the coefficients of the x_i, ($i\in G$) from (7) (or a reduced value as seen below). We also assume that the c_j, a_j

and b are integers. The c_j in (9) are the c_j in (8) multiplied by D. We develop the congruence in (9) by finding the equation in (8) (including the objective function) that has any constant (i. e., considering d, c_j, α_j, and α_0), written in fractional form, having the property that its numerator and denominator are relatively prime. In our case the denominator is D. Gomory [9] shows that if we find a constant with this property, then the congruence taken from the equation where the constant appears generates the congruences developed from the other equations. This is due to the group properties of the fractional parts of the constants in (8). The cutting plane developed from the congruence represents the best cutting plane, since any integer solution to the congruence produces integer solutions for the $x_i (i \in G)$ in (8).

While the search for a constant with the property that numerator and denominator are relatively prime may appear to be tedious, we have developed a simple and rapid way to find the equation containing the constant. We use the revised simplex method to obtain the linear programming solution to (7). Thus, the inverse matrix is available to develop the constants in (8). In solving (7), we carry along the value of D as the product of the pivot elements. In an obvious way the desired equation is found by finding the greatest common divisor of the non-zero numerators of each of the rows of the inverse matrix. We stop when the greatest common divisor is unity. If none of the rows

produce a greatest common divisor of unity, we select the row with minimum greatest common divisor. The greatest common divisor of the greatest common divisors found for each row also divides D. Accordingly, we then produce a smaller value of D by dividing. We use an enlarged inverse matrix to consider the objective function also.

Before entering the above process we first examine the α_0 and d using the Euclidean algorithm. We stop when we reach a fractional constant having unity as the greatest common divisor of the numerator and denominator. If the required row is not found, then we utilize the inverse matrix as above.

We use as an example the problem given in [2], where it illustrates Gomory's algorithm.

$$\begin{aligned} \min \quad & 4x_1 + 5x_2 \\ \text{when} \quad & 3x_1 + x_2 - x_3 = 2 \\ & x_1 + 4x_2 - x_4 = 5 \\ & 3x_1 + 2x_2 - x_5 = 7 \\ & x_j \geq 0 \text{ and integer.} \end{aligned} \tag{10}$$

The continuous solution gives us

$$\begin{aligned} \min \quad & + \frac{112}{10} + \frac{7}{10} x_4 + \frac{11}{10} x_5 \\ & x_1 + \frac{2}{10} x_4 - \frac{4}{10} x_5 = \frac{18}{10} \\ & x_2 - \frac{3}{10} x_4 + \frac{1}{10} x_5 = \frac{8}{10} \\ & x_3 + \frac{3}{10} x_4 - \frac{11}{10} x_5 = \frac{42}{10} \end{aligned} \tag{11}$$

where $D = 10$.

Considering $d = \frac{112}{10}$, $\alpha_0 = (\frac{18}{10}, \frac{8}{10}, \frac{42}{10})$, we see that none have numerator and denominator relatively prime. The inverse matrix is

$$\begin{bmatrix} 0 & \frac{-2}{10} & \frac{4}{10} & 0 \\ 0 & \frac{3}{10} & \frac{-1}{10} & 0 \\ -1 & \frac{-3}{10} & \frac{11}{10} & 0 \\ 0 & \frac{-7}{10} & \frac{-11}{10} & 1 \end{bmatrix}$$

where the bottom row produces the coefficients in the objective function. In the first row we consider the greatest common divisor of $(-2, 4)$, which is 2. The first row produces the first constraint equation in (11). In the second row we consider the greatest common divisor of $(3, -1)$, which is unity. Thus, the required congruence may be obtained from

the second constraint in (11). Similarly, the required congruence may be obtained from the third constraint and the objective function. Using the second constraint, we obtain the congruence

$$7x_4 + x_5 \equiv 8 \pmod{10}.$$

The equivalent knapsack problem becomes

$$\text{minimize } 7x_4 + 11x_5$$

$$\text{when } 7x_4 + x_5 \equiv 8 \pmod{10}$$

$$x_4,\ x_5 \geq 0 \text{ and integer.}$$

In general, once the required equation in (8) is found; e. g. ,

$$x_i + \sum_{j \in \overline{G}} \alpha_{ij} x_j = \alpha_{i0} \quad \text{for a single } i \in G$$

(or for the objective function), the congruence is found as in Gomory [8] as

$$\sum_{j \in \overline{G}} \overline{\alpha}_{ij} x_j \equiv \overline{\alpha}_{i0} \pmod{1}$$

where $\overline{\alpha}_{ij}$ is the fractional part of α_{ij}. Multiplying through by D, we obtain

$$\sum_{j \in \overline{G}} a_j x_j \equiv b \pmod{D}$$

where $a_j = \overline{\alpha}_{ij} D, \quad b = \overline{\alpha}_{i0} D.$

We are interested in the solution to the problem

$$\min \sum_{j=1}^{n} c_j x_j$$

$$\text{when } \sum_{j=1}^{n} a_j x_j \equiv b \pmod{D}.$$

$$x_j \geq 0 \text{ and integer.}$$

We can assume that no value of a_j appears more than once (if not, we can select the one with smallest c_j value).

Writing

$$F(y) = \min \left[\sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} a_j x_j \equiv y \pmod{D} \right]$$

We obtain the recursion as in [7]

$$F(y) = \min (c_j + F(y-a_j)) \tag{12}$$

with $F(0) = 0$. The arguments of F are taken modulo D.

We can write (12) as

$$F(y) = c_r + \min_j (c_j - c_r + F(y-a_j)) \tag{13}$$

where $c_r + \min_j c_j$. Equation (13) has an immediate solution for $y=a_r$, and we obtain $F(a_r) = c_r$. We then write (13) as

$$F(y-a_r) = c_r + \min_j (c_j - c_r + F(y-a_r - a_j)). \tag{14}$$

We substitute (14) for the $F(y-a_r)$ term on the right side of (13) and readily produce another solution to some $F(a_j)$. We stop when

we have achieved a solution for F(b). We also keep track of the variable j, which produces each solution, and then back-track to find the x_j. This entire procedure is contained in the following algorithm:

1. List the values of the problem as follows:

1	2	3	...	n
c_1	c_2	c_3	...	c_n
a_1	a_2	a_3	...	a_n

Go to 2.

2. Given the list

1	2	3	...	n	t_1	t_2	...	t_s
c_1	c_2	c_3	...	c_n	c'_{t_1}	c'_{t_2}	...	c'_{t_s}
a_1	a_2	a_3	...	a_n	a'_{t_1}	a'_{t_2}	...	c'_{t_s}

find $c_r = \min c_j$ for all unmarked columns. Set $F(a_r) = c_r$ and $I(a_r) = r$, where a_r and r are the corresponding elements to c_r in the column. If $a_r = b$, go to 4. Otherwise, mark the column if it is one of the first n. Go to 3.

3. Add columns to the list, if possible, as follows:

Calculate $a'_t \equiv a_r + a_t \pmod{D}$ and $c'_t = c_r + c_t$ for $t=1,\ldots,n$.

Add a column headed by t if:

(a) $F(a'_t)$ has not been found.

(b) a'_t is not on the list.

(c) a'_t is on the list and has a corresponding c_t value that is larger than c'_t.

Delete columns (beyond the first n) having elements a'_t

(d) after $F(a'_t)$ is found.

(e) after a new a'_t is found as in 3(c).

Go to 2.

4. The problem is solved with solution F(b). The values of the variables are found as follows:

(a) Set $x_j = 0$, $j=1,\ldots,n$ and $y=b$. Go to 4(b).

(b) Look up $I(y) = j$. Set $x_j = x_j + 1$. Go to 4(c).

(c) Set $y \equiv y - a_j \pmod{D}$. If $y=0$, go to 5. Otherwise, go to 4(b).

5. End. The final x_j values are the required ones.

This completes the algorithm. If solutions are desired for all values of y, we continue the algorithm instead of stopping when b is reached. In addition, the algorithm may be changed into a single-pass algorithm, as in the following sections.

In the example above,

$$\text{Min} \quad 7x_4 + 11x_5$$

$$\text{when} \quad 7x_4 + x_5 \equiv 8 \pmod{10}$$

$$x_4,\ x_5 \geq 0 \text{ and integer,}$$

we list

4	5
7	11
7	1

Thus, $7 = \min c_j$, $F(7) = 7$, $I(7) = 4$. The list becomes (marking the first column)

4	5	4	5
7	11	14	18
7	1	4	8
*			

Thus, $F(1) = 11$, $I(1) = 5$. The list becomes

4	5	4	5	5
7	11	14	18	22
7	1	4	8	2
*	*			

$F(4) = 14$, $I(4) = 4$. The list becomes

4	5	4	5	5
7	11	18	22	25
7	1	8	2	5
*	*			

$F(8) = 18$, $I(8) = 4$. We stop.

Backtracking, $I(8) = 4$, $x_4 = 1$,

$I(1) = 5$, $x_5 = 1$, end.

The solution is $F(8) = 18$, $x_4 = 1$, $x_5 = 1$.

We have used the above procedure to solve integer programs arising as covering problems. The method usually produces an immediate integer solution. If the knapsack solution does not produce feasible (positive) integers to the problem, we have always been successful after adding a single cut. This was done on approximately 25 problems where there was a maximum of 2000 variables and 150 equations.

3. An Algorithm for Integer Solutions to Linear Programs

In this section, we present an algorithm for the solution of integer linear programs where the variables have upper bounds. The continuous linear programming solution is found first. If the continuous solution is fractional, we develop a linear congruence that is added as a constraint. The optimal solution is then found after a dynamic programming formulation of the problem is made.

We are interested in the integer programming problem: find $X = (x_1, \ldots, x_n)$ that

$$\begin{aligned} &\text{minimizes} && CX \\ &\text{when} && AX = B \\ & && 0 \le x_j \le b_j, \quad x_j \text{ integer}, \quad j=1, \ldots, n, \end{aligned} \tag{15}$$

where C is an n-vector, B is an m-vector, A is an mxn matrix, and the b_j are integers (any or all of the b_j may be infinite). We

assume that the components of C, B, and the elements of A are integers.

The solution of (15) includes the case where the x_j are restricted to being either zero or one.

We use a bounded variable linear programming technique as in [3] to obtain the equivalent problem:

$$\begin{aligned} &\text{minimize} \quad d + \sum_{j\in\bar{G}} c_j x_j \\ &X_G + \sum_{j\in\bar{G}} \alpha_j x_j = \alpha_0, \qquad (16) \\ &0 \le x_j \le b_j, \quad x_j \text{ integer}, \quad j=1,\ldots,n, \end{aligned}$$

where the vector $X_G = \{x_i \mid i\in G\}$, G is the set of indices of the basic variables, and $\bar{G}$ is the set of indices of the non-basic variables. Further, some of the non-basic variables may be at their upper bounds in the continuous solution of (15). For those variables we would have their corresponding c_j values as non-positive. We also have $c_j \ge 0$ for non-basic variables that are at the zero value. For ease of computation, we make the transformation $x_j' = b_j - x_j$ for those non-basic variables at their upper bound. Thus, we may assume a form like (16) with all $c_j \ge 0$ and all non-basic variables at zero values. The vectors α_j ($j=0$ and $j\in\bar{G}$) are column vectors. The components of α_0 are non-negative. If α_0 is all integer, then $x_j = 0$ ($j\in\bar{G}$), $X_G = \alpha_0$ is an optimal solution (if any of the x_j, $j\in\bar{G}$ are really the x_j' under

the transformation, we would have $x_j = b_j$). If any of the α_0 are fractional, problem (16) is converted to the problem:

$$\begin{aligned} &\text{minimize} && \sum_{j\in\bar{G}} c_j x_j \\ &\text{when} && \sum_{j\in\bar{G}} \alpha_j x_j \le \alpha_0 \\ & && \sum_{j\in\bar{G}} a_j x_j \equiv b \pmod{D} \\ & && 0 \le x_j \le b_j,\ x_j \text{ integer},\ j\in\bar{G} \end{aligned} \tag{17}$$

The inequality constraints in (17) are obtained from (16) by dropping the x_i, $i\in G$. D is the determinant of the coefficients of the x_i, $i\in G$ from (15). The value of D may be found as the product of the pivot elements during the simplex procedure for obtaining the continuous solution of (15). The linear congruence in (17) is a single congruence where the a_j and b are integers. The congruence results from the requirement that all x_i, $i\in G$, be integers. We develop the congruence, as in the previous algorithm, by finding a constant among d, c_j, α_j, and α_0, written in fractional form as m/D, having the property that m and D are relatively prime. We previously showed how to find the equation of (16) with the required constant. We then develop the congruence from the equation. Most linear programming problems have certain group properties [9], so that any integer solution of the congruence produces integer values for the x_i, $i\in G$.

We proceed to develop the dynamic programming formulation of (17). Writing

$$F(x, \alpha, \beta) = \min\left[\sum_{j\epsilon\overline{G}} c_j x_j \;\middle|\; \sum_{j\epsilon\overline{G}} \alpha_j x_j \leq \alpha, \quad \sum_{j\epsilon\overline{G}} a_j x_j \equiv x \pmod{D}, \quad 0 \leq e_j x_j \leq \beta \right]$$

we obtain the dynamic programming recursion

$$\begin{aligned} F(x, \alpha, \beta) &= \min_{j\epsilon\overline{G}} \left(c_j + F(x-a_j, \ \alpha-\alpha_j, \ \beta-e_j)\right) \\ F(0, \alpha, \beta) &= 0 \quad \text{for} \quad \alpha \geq 0, \quad \beta \geq 0. \end{aligned} \tag{18}$$

The x argument of F is taken modulo D. The vector e_j is composed of a unit element with the other elements zero. The unit element corresponds to the unit coefficient of x_j in $x_j \leq b_j$. We also assume that only feasible values of x, α, and β are used in (18).

One immediate solution to (18) can be obtained by finding $c_r = \min c_j$. This produces $F(a_r, \alpha_r, e_r) = c_r$. We then replace x by $x-a_r$, α by $\alpha-\alpha_r$, and β by $\beta-e_r$ in (18), and we substitute the result for the $F(x-a_r, \alpha-\alpha_r, \beta-e_r)$ term on the right side of (18). We then can produce another immediate solution. We continue this way until $F(b, \alpha, \beta)$ is found where $0 \leq \alpha_0 - \alpha \leq b_G$ and $\beta \leq b_{\overline{G}}$ where the vector $b_G = \{b_i | i\epsilon G\}$ and the vector $b_{\overline{G}} = \{b_i | i\epsilon\overline{G}\}$. This entire procedure is contained in the following algorithm:

1. Suppose the indices in $\overline{G}$ are $(1, 2, \ldots, m)$, list the values of the problem as

1	2	3	...	m
c_1	c_2	c_3	...	c_m
α_1	α_2	α_3	...	α_m
a_1	a_2	a_3	...	a_m
$x_j = 0$				

Go to 2.

2. Given the list, find $c_r = \min c_j$ for all unmarked columns in all sections. If $a_r = b$ and $0 \leq \alpha_0 - \alpha_r \leq b_G$, go to 4. Otherwise, mark the column and go to 3.

3. Add a new section of columns to the list, if possible, as follows:

   a) calculate $c'_j = c_r + c_j$, $\alpha'_j = \alpha_r + \alpha_j$, $a'_j \equiv a_r + a_j \pmod{D}$ for all values $j \in \overline{G}$ (i.e., values j, c_j, α_j, a_j are taken from the list in step 1). where $x_j < b_j$ for $j \neq r$ and where $x_r + 1 < b_r$ for $j=r$ for the section containing the newly marked column.

   b) add the column headed by j in the new section with values c'_j, α'_j and a'_j.

   c) underneath the section added write the x_j values from the section containing the newly marked column. Increase x_r by one for the section.

When columns in new sections become marked, they may be deleted after the calculations are made and the new section is added to the list. A new column may already be on the list. It need not be added if the x_j values in corresponding sections would coincide (after increasing the x_j values by one for the j heading values of the two columns). Go to 2.

4. The integer problem is solved with objective function value $d + c_r$. The values of the basic variables in (16) are $X_G = \alpha_0 - \alpha_r$. The values of the non-basic variables are found below the section where $a_r = b$ appears; increase x_r by one.

This completes the algorithm. An integer solution is sure to be found because all integer solutions to the congruence in (17) are systematically produced in order of increasing objective function value. The algorithm may be simplified by calculating α'_j only when $a'_j = b$ by using the proper x_j values and the α_j from step 1. Furthermore, if any of the components of X_G in 4 are fractional, then we have a multi-dimensional group. We simply add another congruence and treat a_j and b in (17) as vectors and continue the algorithm in step 2.

Example: We take the problem given in [1]

$$\begin{aligned}
\text{minimize} \quad & 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5 \\
\text{when} \quad & x_1 - 3x_2 + 5x_3 + x_4 - 4x_5 \geq 2 \\
& -2x_1 + 6x_2 - 3x_3 - 2x_4 + 2x_5 \geq 0 \\
& - x_2 + 2x_3 - x_4 - x_5 \geq 1 \\
& 0 \leq x_j \leq 1, \quad x_j \text{ integer}, \quad j = 1, \ldots, 5.
\end{aligned}$$

We introduce surplus variables $x_6 \geq 0$, $x_7 \geq 0$, and $x_8 \geq 0$ (i.e., with ∞ as upper bounds) and find the continuous solution and equivalent problem:

$$\begin{aligned}
\text{minimize} \quad & 9 + \tfrac{93}{9}x_1 + \tfrac{156}{9}x_4 + \tfrac{42}{9}x_5 + \tfrac{24}{9}x_7 + \tfrac{137}{9}x_8 \\
& -\tfrac{7}{9}x_1 - \tfrac{28}{9}x_4 + \tfrac{13}{9}x_5 + x_6 + \tfrac{1}{9}x_7 - \tfrac{1}{3}x_8 = \tfrac{1}{3} \\
& -\tfrac{2}{9}x_1 + x_3 - \tfrac{8}{9}x_4 - \tfrac{4}{9}x_5 - \tfrac{1}{9}x_7 - \tfrac{2}{3}x_8 = \tfrac{2}{3} \\
& -\tfrac{4}{9}x_1 + x_2 - \tfrac{7}{9}x_4 + \tfrac{1}{9}x_5 - \tfrac{2}{9}x_7 + \tfrac{1}{3}x_8 = \tfrac{1}{3} \qquad (19) \\
& 0 \leq x_j \leq 1, \quad x_j \text{ integer}, \quad j = 1, \ldots, 5 \\
& 0 \leq x_j, \quad j = 6, 7, 8.
\end{aligned}$$

We have $D = 9$. The second equation produces the congruence

$$7x_1 + x_4 + 5x_5 + 8x_7 + 3x_8 \equiv 6 \pmod 9$$

which is added as a constraint. Multiplying through by 9 where applicable, we list

	1	4	5	7	8
9	93	156	42	24	137
3	-7	-28	13	1	-3
6	-2	-8	-4	-1	-6
3	-4	-7	1	-2	3
6	7	1	5	8	3
			*	*	
	$x_i = 0$				

(20)

For convenience we include at the left of the list the objective function value, the right-hand sides (multiplied by 9) for the equalities in (19), and the right-hand side for the congruence. We have $24 = \min c_j$; i.e., $c_r = 24$, with $r = 7$. We add the section

1	4	5	7	8
117	180	66	48	161
-6	-27	14	2	-2
-3	-9	-5	-2	-7
-6	-9	-1	-4	1
6	0	4	7	2
		*	*	
$x_7 = 1$				

(21)

We mark the 7 column in (20). We have $\min c_j = 42$ from (20); we add the section

1	4	8
135	198	178
6	-15	10
-6	-12	-10
-3	-6	4
3	6	8
$x_5 = 1$		

(22)

We need not add a column headed by 7 in (22) because it would duplicate the 5 column in (21) (the use of either column would require $x_5 = 1$ and $x_7 = 1$). We do not add a 5 column in (22) since x_5 would be at its upper bound in (22). We mark the 5 column in (20). We have $\min c_j = 48$ in (21). We add the section

1	4	5	7	8
141	204	90	72	185
-5	-26	15	3	-1
-4	-10	-6	-3	-8
-8	-11	-3	-6	-1
5	8	3	6	1
$x_7 = 2$				

(23)

We mark the 7 column in (21). We have $\min c_j = 66$ from (21). We add the section

1	4	8
159	222	203
-7	-14	11
-7	-13	-11
-5	-8	2
2	5	7
$x_1 = 1, \quad x_5 = 1$		

(24)

We mark the 5 column in (21). The solution is now apparent in the 7 column in (23). We have the optimal solution $x_7 = 3$, $x_6 = (3 - 3)/9 = 0$, $x_3 = (6 + 3)/9 = 1$, $x_2 = (3 + 6)/9 = 1$; the objective value is $9 + 72/9 = 17$.

4. The Solution of Integer Programs by Bounded Variable Techniques

In this section, the integer programming problem is solved by linear programming using upper and lower bounds on the variables. The bounds are changed systematically until the optimal integer solution is produced. The method results in a branching process that requires minimum excess computer storage over that required to solve the linear programming problem. In addition, dynamic programming enumeration techniques are used to speed the calculations.

The basic method may be applied to problems where not all the variables are integer restricted.

The branch and bound method of Land and Doig [10] is to solve (15) first as a linear program to obtain the continuous solution $(z^0, x_1^0, \ldots, x_n^0)$. If the continuous solution is all integer, the problem is solved. Otherwise, the value of the objective function z^0 provides a lower bound on the value of the optimal integer-solution; some variable x_k is selected whose value x_k^0 is not an integer. Thus, integer x_k must satisfy either $x_k \leq [x_k^0]$ or $x_k \geq [x_k^0] + 1$, where $[x_k^0]$ is the greatest integer less than x_k^0. The procedure initially involves the solution of two problems: the linear program min z subject to $x_k = [x_k^0]$, in one case, and $x_k = [x_k^0] + 1$, in the other case, added to the constraints of (15).

Suppose the solutions are $(z^1, x_1^1, \ldots, [x_k^0], \ldots x_n^1)$ and $(z^2, x_1^2, \ldots, [x_k^0] + 1, \ldots, x_n^2)$ with $z^1, z^2 \geq z^0$. The problem solution with smaller z^1 or z^2 is investigated first (while the other problem is stored for possible later use). Suppose it is z^1. If the z^1 solution problem is all integer, then (15) is solved. Otherwise, some variable x_r is selected whose value x_r^1 is not an integer. Thus, three new problems are solved: the linear programs min z subject to $x_k = [x_k^0]$, $x_r = [x_r^1]$, in the first case, $x_k = [x_k^0]$, $x_r = [x_r^1] + 1$, in the second case, and $x_k = [x_k^0] - 1$,

in the third case, added to the constraints of (15) (assuming $[x_k^0] \neq 0$).

The process is then repeated. There are four problem solutions that are investigated. The method continues in the above fashion until an integer solution is achieved.

This entire procedure enumerates possibilities for the solution to (15) in a directed tree. The rooted node corresponds to z^0 and has directed branches to nodes corresponding to the solution of the linear program (15) with x_k set at specific integer values, each representing a single branch. From each node the same procedure is repeated. The Land and Doig method develops the tree by branching from the terminal node that has minimum objective value. This will then limit the potentially large size of the tree.

The Land and Doig method works well when there are few variables in a problem. The method does not work well when the number of variables is large due to the possibly immense storage requirements for saving the problems that represent the terminal nodes.

In this section, we develop a different branching procedure that requires a minimum of excess storage beyond that required to solve the linear programming problem. In addition, dynamic programming enumeration techniques are used to reduce the number of branchings required.

We solve (15) as a linear program to obtain the continuous solution. For a new branching procedure several observations are possible first:

(a) The Land and Doig method may be simplified by solving the two problems: min z subject to $x_k \leq [x_k^0]$, in one case, and $x_k \geq [x_k^0] + 1$, in the other case, added to the constraints of (15). This is done also for any terminal node. Thus, any terminal node would have a maximum of two branches leading to the next level of the tree. This procedure tends to reduce the number of nodes that we need to investigate.

(b) The procedure may be further simplified when a feasible integer solution is obtained at any node. The objective value $z = x_0$ represents an upper bound to the optimal z. A good strategy is to investigate a single branch of the tree until $[z^k] + 1 \geq x_0$. We then backtrack in the tree until a node is reached that has had only one branching. This is achieved in the problem by removing the bounds beyond the node and solving the resultant linear program. Then the second branch is made (i.e., in back-tracking we reach a node that has only $x_k \geq [x_k^r] + 1$ as a branch. We remove all bounds beyond the node and solve the linear program starting from the latest transformation of the equations. We then impose $x_k \leq [x_k^r]$ and continue on a single

branch). This procedure has the advantage of developing only one level of the tree and storing only the current bounds. If any feasible integer solution is found with objective value less than x_0, we use the new solution as an upper bound to the optimal value of z. Solution is achieved when it is impossible to branch any further. The current upper bound solution is then optimal.

(c) Feasible integer solutions may be found using a dynamic programming enumeration similar to that given in the previous section. This can be done at any node. In performing the enumeration it can be noted when x_0 will be surpassed. The enumeration can then be stopped and the back-tracking procedure in (b) instituted.

(d) The enumeration procedure can be instituted at the root node, thus producing the optimal solution. However, if D (the product of the pivot elements in solving (15) by the simplex procedure [3] is large, then the dynamic programming enumeration may require excess storage. However, the value of D tends to reduce as more bounds are made. Thus, the dynamic programming enumeration will require less storage as higher levels in the tree are reached.

(e) The selection of the non-integer variable in the branching may

be critical in reducing the number of nodes. When D is large, we use the criteria of [4]. When D is small enough, we use a dynamic programming iteration. It is possible to stop the enumeration before completion and obtain infeasible integers for the variables. We have then branched on the variable that is most negative.

We have used the two-phase simplex method to solve (15) together with the bounded variable technique in [3]. Thus, we maintain m equations and have a solution where some variables will be basic, some will be zero, and some will be at their upper bound. If all b_j are infinite, the method reverts to the usual one. In the branching procedure some variables will have non-zero lower bounds.

We are interested in finding the solution criteria to (15) with bounds given by $0 \le \bar{a}_j \le x_j \le \bar{b}_j \le b_j$, $j = 1, \ldots, n$. Suppose we achieve solution values for the x_j, satisfying the bounds plus the constraint equations, and have obtained the following canonical form [3] from the constraint equations in (15):

$$\begin{aligned} x_i + \sum_{j=m+1}^{n} a_{ij} x_j &= d_i \qquad i = 1, \ldots, m \\ -z + \sum_{j=m+1}^{n} c_j x_j &= -z_0. \end{aligned} \tag{25}$$

The variables x_i, $i = 1, \ldots, m$, are the basic variables, and the variables x_j, $j = m + 1, \ldots, n$, are the non-basic variables. We have the following:

Theorem: Any values of x_j, $j = 1, \ldots, n$ satisfying the constraint equations in (25) and $0 \leq \bar{a}_j \leq x_j \leq \bar{b}_j \leq b_j$, $j = 1, \ldots, n$ are an optimal solution for min z, with objective value z_0, if $c_j \geq 0$ for variables at their lower bound $\bar{a}_j$ and $c_j \leq 0$ for variables at their upper bound $\bar{b}_j$.

Proof: Any increase in the variables at their lower bound or any decrease in the variables at their upper bound can only increase z.

The theorem produces the conditions for optimality. To improve a feasible solution that is not optimal, we can increase a non-basic variable x_j at its lower bound when $c_j < 0$, or can decrease a non-basic variable x_j at its upper bound when $c_j > 0$. Thus, when a feasible solution to the linear programming problem with upper and lower bounds can be found, we can use a variant of the usual simplex procedure to achieve optimality.

After solving (15) as a linear program, we impose the condition $x_k \leq [x_k^0]$ or $x_k \geq [x_k^0] + 1$ for a basic variable x_k with its value x_k^0 as fractional. The solution for the linear program is then infeasible. We next place x_k at either the upper bound $[x_k^0]$ or the

lower bound $[x_k^0]$ + 1; add an artificial variable equal to the amount of the infeasibility produced ($x_k^0 - [x_k^0]$, in the first case, or $[x_k^0] + 1 - x_k^0$, in the second case); and solve the new bounded variable problem by the two-phase procedure. If we obtain a solution to the new problem that is fractional, we impose the additional condition $x_r \leq [x_r^1]$ or $x_r \geq [x_r^1] + 1$ for a basic variable x_r with its value x_r^1 as fractional. We then solve the new problem maintaining the constraint on x_k. We continue this process until it is necessary to backtrack, which is done by simply removing the bounds on the required variables and solving the linear program to return to a previous problem. Note that bounds on the same variable may appear again; the initial bounds will automatically be satisfied.

This entire procedure enumerates possibilities for the solution to (15) in a directed tree. The rooted node corresponds to z^0. We only investigate one branch to the next node corresponding to the solution of (15) with, say, $x_k \geq [x_k^0] + 1$ representing the branch. From each node the same procedure is repeated, investigating one branch until we backtrack to a previous node that has had only one branching. For example, we would branch to a node corresponding to the solution of (15) with $x_k \leq [x_k^0]$ representing the branch when we backtrack to the rooted node. Any further backtracking to the

rooted node would end the problem. This procedure is contained in the following algorithm:

Define $S(x_0, x_1, \ldots, x_n)$ as the current upper bound solution, which represents a feasible integer solution to (15) with objective value x_0.

Define $I(L)$ as the index of the L^{th} bounded variable.

Define $N(L)$ to show the number of branchings for the L^{th} bounded variable (Initially $N(L) = 0$).

Define $B(L)$ to show the bound on the L^{th} bounded variable.

Define $G(L) = 0$ to indicate the L^{th} bounded variable is $\leq B(L)$ and $G(L) = 1$ to indicate the L^{th} bounded variable is $\geq B(L)$.

1. Set $L = 0$ and take $S(x_0, x_1, \ldots, x_n)$ as a feasible integer solution to (15). If none is apparent, take x_0 as infinite. Solve (15) as a linear program. If the solution is all integer, the problem is solved. Otherwise, go to 2, maintaining the canonical form of the solution to (15).

2. Set $L = L + 1$. Go to 2(a).

   (a) Select a basic variable that is fractional. Suppose the variable selected is x_k with value x_k^0, set $I(L) = k$, $N(L) = 1$ and go to 2(b) or 2(c).

   (b) Set $B(L) = [x_k^0]$ and $G(L) = 0$. Go to 3.

   (c) Set $B(L) = [x_k^0] + 1$ and $G(L) = 1$. Go to 3.

3. Solve the linear program using the maintained canonical form with the new bounds on the variable x_k where $k = I(L)$. One of the cases holds:

   (a) If the solution produces an objective value z with $[z] + 1 \geq x_0$ go to 4.

   (b) If the solution produces an objective value z with $[z] + 1 < x_0$ and the solution is all integer, redefine $S(x_0, x_1, \ldots, x_n)$ as a new feasible integer solution where $x_0 = z$, $x_1, \ldots, x_n$ is the new solution. Go to 4.

   (c) If the solution produces an objective value z with $[z] + 1 < x_0$ and the solution is fractional, go to 2.

   (d) If the problem has an infeasible solution, go to 4.

4. Set $LL = L$ and go to 4(a).

   (a) If $N(L) = 2$, go to 4(b). Otherwise, $N(L) = 1$; go to 4(c).

   (b) If $L = 1$, the feasible solution $S(x_0, x_1, \ldots, x_n)$ is optimal. Stop. Otherwise, set $N(L) = 0$, $L = L - 1$, and go to 4(a).

   (c) Solve the linear programming problem starting from the current canonical form with the bounds $B(L)$, $B(L + 1), \ldots, B(LL)$ removed. If $K = I(L)$ and x_k

does not become a basic variable, then the solution is non-unique; x_k is then made a basic variable. Go to 4(d).

(d) If G(L) = 0, set B(L) = B(L) + 1, N(L) = -2, and go to 3.

Otherwise, G(L) = 1; set B(L) = B(L) - 1, N(L) = 2, and go to 3.

This completes the branching part of the algorithm. Good selection rules for which basic variable to use in 2(a) and the choice of 2(b) or 2(c) are given in [4]. The dynamic programming enumeration, which speeds the convergence in the algorithm, is carried out after the linear programming solutions are found in steps 1 or 3.

We can illustrate the basic branching method, as given above, where the variables are arbitrarily picked for branching, and the branching is always taken with a greater than or equal sign first, for a problem [2] used to illustrate the Land and Doig procedure:

$$\begin{aligned} \text{Min} \quad & 4x_1 + 5x_2 \\ \text{when} \quad & 3x_1 + x_2 - x_3 = 2 \\ & x_1 + 4x_2 - x_4 = 5 \\ & 3x_1 + 2x_2 - x_5 = 7 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \text{ and integer.} \end{aligned} \tag{26}$$

The continuous solution produces the canonical form

$$\begin{aligned} x_1 + \frac{2}{10}x_4 - \frac{4}{10}x_5 &= \frac{18}{10} \\ x_2 - \frac{3}{10}x_4 + \frac{1}{10}x_5 &= \frac{8}{10} \\ x_3 + \frac{3}{10}x_4 - \frac{11}{10}x_5 &= \frac{42}{10} \\ -z + \frac{7}{10}x_4 + \frac{11}{10}x_5 &= -\frac{112}{10} \end{aligned} \tag{27}$$

Since the solution is fractional, we arbitrarily select the bound $x_1 \geq 2$ added to (27). We then obtain the canonical form

$$\begin{aligned} x_2 + \frac{1}{4}x_1 - \frac{1}{4}x_4 &= \frac{5}{4} \\ x_3 - \frac{11}{4}x_1 - \frac{1}{4}x_4 &= -\frac{3}{4} \\ x_5 - \frac{5}{2}x_1 - \frac{1}{2}x_4 &= -\frac{9}{2} \\ -z + \frac{11}{4}x_1 + \frac{5}{4}x_4 &= -\frac{25}{4} \end{aligned} \tag{28}$$

where $(z, x_1, x_2, x_3, x_4, x_5) = (47/4,\ 2,\ 3/4,\ 19/4,\ 0,\ 1/2)$.

We arbitrarily select $x_2 \geq 1$ and solve (28) with $x_1 \geq 2$ and $x_2 \geq 1$ to obtain the canonical form

$$\begin{aligned} x_3 - 3x_1 - x_2 &= -2 \\ x_4 - x_1 - 4x_2 &= -5 \\ x_5 - 3x_1 - 2x_2 &= -7 \\ -z + 4x_1 + 5x_2 &= 0 \end{aligned} \tag{29}$$

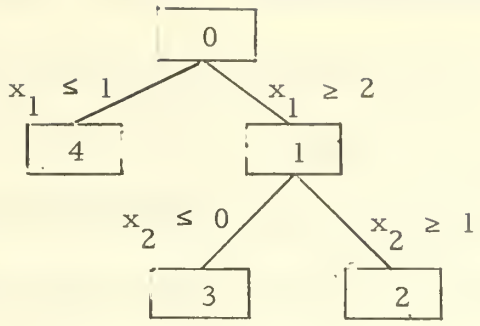
where $(z, x_1, x_2, x_3, x_4, x_5) = (13, 2, 1, 5, 1, 1)$, which is all integer and becomes the solution vector $S(x_0, x_1, \ldots, x_n)$. We then remove the bound $x_2 \geq 1$ and apply the simplex procedure on (29) (e.g., z may be decreased by decreasing x_2) and obtain (28) again. We then impose the bound $x_2 \leq 0$ on (28) in addition to $x_1 \geq 2$ and obtain the canonical form

$$\begin{aligned} x_1 + 4x_2 - x_4 &= 5 \\ x_3 + 11x_2 - 3x_4 &= 13 \\ x_5 + 10x_2 - 3x_4 &= 8 \\ -z - 11x_2 + 4x_4 &= -20 \end{aligned} \tag{30}$$

where the solution is $(20, 5, 0, 13, 0, 8)$. Since $z > 13$, we remove the bounds $x_2 \leq 0$ and $x_1 \geq 2$ and solve (30) to obtain (27) again. We take $x_1 \leq 1$ and obtain the canonical form

$$\begin{aligned} x_2 + \frac{3}{2}x_1 - \frac{1}{2}x_5 &= \frac{7}{2} \\ x_3 - \frac{3}{2}x_1 - \frac{1}{2}x_5 &= \frac{7}{2} \\ x_4 + 5x_1 - 2x_5 &= 9 \\ -z - \frac{7}{2}x_1 + \frac{5}{2}x_5 &= \frac{-37}{2} \end{aligned} \tag{31}$$

where $z, x_1, x_2, x_3, x_4, x_5) = (14, 1, 2, 3, 4, 0)$. Since $z > 13$, the problem is solved with solution $z = 13$, $x_1 = 2$, $x_2 = 1$, $x_3 = 5$, $x_4 = 1$, $x_5 = 1$. The procedure is exhibited in the following tree.



Node 0: (112/10, 18/10, 8/10, 42/10, 0, 0)

Node 1: (47/4, 2, 3/4, 19/4, 0, 1/2)

Node 2: (13, 2, 1, 5, 1, 1)

Node 3: (20, 5, 0, 13, 0, 8)

Node 4: (14, 1, 2, 3, 4, 0)

The advantages of this basic branching algorithm are apparent over the Land and Doig procedure even in this small example. As shown in [2], the usual procedure requires 5 nodes. Here we need only 4 nodes. In addition, we can obtain feasible integer solutions more readily; e.g., at node 3, because $x_1 \geq 2$ allowed x_1 to become a basic variable again with a value that produced integers. Further, no additional problems are generated except that the canonical form from the original continuous solution is varied depending on the bounds on the variables. Each problem leads into the next in a natural fashion.

The branching method outlined above will work well if feasible integer solutions can be found rapidly. We can achieve this by instituting

a dynamic programming enumeration, similar to that described in the previous section, after a fractional solution is obtained at any node. The equations at any stage are given by (25). We select one equation in (25) and develop a linear congruence as in [9]. Many problems have systems of equations, with certain group properties [9], so that any integer solution to the congruence produces integer solutions for all variables. In problems without this group property (i.e., multidimensional groups), we simply add additional congruences when necessary. Suppose the congruence obtained from (25) is

$$\sum_{i=m+1}^{n} a_j x_j \equiv d \pmod{D}. \tag{32}$$

We then add (32) as a constraint to (25) and enumerate all integer solutions to (32) in a dynamic programming format to minimize z subject to the bounds on the variables. We continue until we achieve feasible integer solutions. The values are then optimal from the form of the enumeration. To simplify the calculation using (26), we make the transformation $y_j = x_j + \bar{a}_j$; we then have $0 \le y_j \le b'_j = \bar{b}_j - \bar{a}_j$. For variables with $c_j < 0$ in the objective function of (25), we make the additional transformation $y'_j = b'_j - y$.

After these transformations a dynamic programming enumeration may be made. Using methods similar to that given in the previous section, we obtain the following algorithm for the solution to (25) under

the transformations above, where we take $c_j \geq 0$, and (32) becomes

$$\sum_{j=1}^{m'} a_j y_j \equiv b \pmod{D}. \tag{33}$$

1. Suppose the y_j indices are $j=1, \ldots, m'$, list the values of the problem as

1	2	...	m'
c_1	c_2	...	$c_{m'}$
a_1	a_2	...	$a_{m'}$
$y_j = 0$			

   Go to 2.

2. Given the list, find $c_r = \min c_j$ for all unmarked columns in all sections. Mark the column. If $a_r = b$, go to 4. Otherwise, go to 3.

3. Add a new section of columns to the list, if possible, as follows:

   (a) Calculate $c'_j = c_r + c_j$, $a'_j \equiv a_r + a_j \pmod{D}$ for all values j on the list in step 1, where $x_j < b'_j$ for $j \neq r$ and where $x_r + 1 < b'_r$ for $j = r$, for the section containing the newly marked column.

   (b) Add the column headed by j in the new section with values c'_j and a'_j.

   (c) Underneath the section write the x_j values from the section containing the newly marked column. Increase x_r by one for the section.

When columns in new sections become marked, they may be deleted after the calculations are made, and the new section is added to the list. A new column may already be on the list. It need not be added if the x_j values in corresponding sections would coincide (after increasing the x_j values by one for the j heading values of the two columns). Go to 2.

4. Calculate the values of the basic variables in (25) using the y_j values under the marked section with y_r increased by one (i.e., the y_j are transformed back to the x_j). We have one of 3 cases:

   (a) If the basic variables have feasible integer solutions, the problem is solved.

   (b) If any basic variable has a fractional value, find another congruence from an equation with fractional value and treat a_j and b as vectors in (33). Go to 2.

   (c) If any integer value of a basic variable is infeasible, go to 2.

This completes the algorithm. As an example, suppose we use the canonical form in (28). The congruence may be taken from the x_2 equation and is

$$x_1 + 3x_4 \equiv 1 \pmod 4. \tag{34}$$

Making the change of variables $y_1 = x_1 + 2$, $y_4 = x_4$, we obtain

$$y_1 + 3y_4 \equiv 3 \pmod 4 \tag{35}$$

from (34) and

$$z = \frac{47}{4} + \frac{11}{4}y_1 + \frac{5}{4}y_4. \tag{36}$$

The enumeration consists in listing the indices for y_1 and y_4 plus the coefficients from (35) and (36) (multiply by 4 in (36)) as

1	4
11	5
1	3
$y_1 = 0$	
$y_4 = 0$	

We obtain a possible solution in the 4 column with values $z = 13$, $y_4 = 1$, and $y_1 = 0$, which produces $x_4 = 1$, $x_1 = 2$. Substituting in (28), we obtain $x_2 = 1$, $x_3 = 5$, $x_5 = 1$. Thus, no further branching would be necessary from node 1. If x_2, x_3, or x_5 would be infeasible with $x_4 = 1$ and $x_1 = 2$, we would continue the enumeration.

In addition, if a solution vector $S(x_0, x_1, \ldots x_n)$ is available and the enumeration would produce an integer $z \geq x_0$, we simply stop the enumeration and backtrack as in step 4 of the basic algorithm.

BIBLIOGRAPHY

1. Balas, E., "Discrete Programming by the Filter Method," Operations Research, Vol. 15 (1967), pp. 915-957.

2. Balinski, M. L., "Integer Programming: Methods, Uses, Computation," Management Science, Vol. 12 (1965), pp. 253-313.

3. Dantzig, G. D., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey (1963).

4. Driebeck, N. J., "A Method for the Solution of Mixed Integer and Nonlinear Programming Problems by Linear Programming Methods," International Symposium on Mathematical Programming, London (1964). (The methods are more readily available in Beale, E. M. L., Mathematical Programming in Practice, Sir Isaac Pitman and Sons, Ltd., London (1968)).

5. Gilmore, P. C. and Gomory, R. E., "A Linear Programming Approach to the Cutting Stock Problem," Operations Research, Vol. 9 (1961), pp. 849-859.

6. Gilmore, P. C. and Gomory, R. E., "A Linear Programming Approach to the Cutting Stock Problem - Part II," Operations Research, Vol. 11 (1963), pp. 863-888.

7. Gilmore, P. C. and Gomory, R. E., "The Theory and Computation of Knapsack Functions," Operations Research, Vol. 14 (1966), pp. 1045-1074.

8. Gomory, R. E., "On the Relation Between Integer and Non-Integer Solutions to Linear Programs," Proc. Nat. Acad. Science, Vol. 53 (1965), pp. 260-265.

9. Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs," in Recent Advances in Mathematical Programming, Graves, R. L. and Wolfe, P. (Eds.), McGraw-Hill, New York, (1963), pp 269-302.

10. Land, A. H. and Doig, A. G., "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol. 28 (1960), pp. 497-520.

11. Shapiro, J. F., "Dynamic Programming Algorithms for the Integer Programming Problem - I: The Integer Programming Problem Viewed as a Knapsack Type Problem," Operations Research, Vol. 16 (1968), pp. 103-121.

12. Shapiro, J. F. and Wagner, H. M., "A Finite Renewal Algorithm for the Knapsack and Turnpike Models," Operations Research, Vol. 15 (1967), pp. 319-341.

DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center (DDC) Cameron Station Alexandria, Virginia 22314	20
2. Library Naval Postgraduate School Monterey, California 93940	2
3. Dr. Harold Greenberg Associate Professor Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	40
4. Dean of Research Naval Postgraduate School Monterey, California	1
5. Library Dept. of Operations Analysis Naval Postgraduate School Monterey, California	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
ON SOLVING INTEGER PROGRAMS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*
Greenberg, Harold

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
May 1968	50	12

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	NPS55GD 8051A
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Foundation Research Program, Naval Postgraduate School

13. ABSTRACT

This report contains new methods of finding integer solutions to linear programming problems. The approaches presented, with illustrative examples, emphasize the use of dynamic programming techniques.

In addition, a new branching scheme is presented that is a natural extension of linear programming methods.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Integer linear programming						
Knapsack problems						
New algorithms						